UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

| APPLICATION NO. | FILING DATE | FIRST NAMED INVENTOR | ATTORNEY DOCKET NO. | CONFIRMATION NO. |
|---|---|---|---|---|
| 10/777,909 | 02/12/2004 | William Preston Alexander III | AUS920030825US1 | 6082 |

35525 7590 03/19/2009

IBM CORP (YA)
C/O YEE & ASSOCIATES PC
P.O. BOX 802333
DALLAS, TX 75380

| EXAMINER | |
|---|---|
| RAMPURIA, SATISH | |

| ART UNIT | PAPER NUMBER |
|---|---|
| 2191 | |

| NOTIFICATION DATE | DELIVERY MODE |
|---|---|
| 03/19/2009 | ELECTRONIC |

**Please find below and/or attached an Office communication concerning this application or proceeding.**

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

ptonotifs@yeeiplaw.com

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

**BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES**

Application Number: 10/777,909
Filing Date: February 12, 2004
Appellant(s): ALEXANDER ET AL.

Gerald H. Glanzman
For Appellant

**EXAMINER'S ANSWER**

This is in response to the appeal brief filed 12/09/2008 appealing from the Office action (Final) mailed 07/22/2008.

**(1) Real Party in Interest**

A statement identifying by name the real party in interest is contained in the brief.

Based on the information supplied by the Appellants, and to the best of Appellants' legal representative's knowledge, the real party in the interest is the assignee, International Business Machines Corporation of Armonk, New York.

**(2) Related Appeals and Interferences**

The examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

**(3) Status of Claims**

The statement of the status of claims contained in the brief is correct and copied below for convenience.

Claims canceled: 2, 3 and 9-24.

Claims withdrawn from consideration but not canceled: None.

Claims pending: 1 and 4-8.

Claims allowed: None.

Claims rejected: 1 and 4-8.

Claims objected to: None.

Claims on Appeal: 1 and 4-8.

**(4) Status of Amendments After Final**

The appellant's statement of the status of amendments after final rejection contained in the brief is correct.

**(5) Summary of Claimed Subject Matter**

The summary of claimed subject matter contained in the brief is correct.

**(6) Grounds of Rejection to be Reviewed on Appeal**

The appellant's statement of the grounds of rejection to be reviewed on appeal is correct.

**(7) Claims Appendix**

The copy of the appealed claims contained in the Appendix to the brief is correct.

**(8) Evidence Relied Upon**

6,338,159 Alexander, III et al. 01-2002

Kazi et al., "JaViz: A client/server Java profiling tool", IBM Systems Journal 39, No. 1, 2000, Pages: 96-117.

**(9) Grounds of Rejection**

The following ground(s) of rejection are applicable to the appealed claims:

***Claim Rejections - 35 USC § 103***

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

Claims 1 and 4-8 rejected under 35 U.S.C. 103(a) as being unpatentable over US Patent No. 6,338,159 to Alexander III et al. (hereinafter, Alexander(1)) in view of document (Title: JaViz: A client/server Java profiling tool) published on 2000 by Kazi et al. (hereinafter, Kazi).

Alexander(1) discloses:

1. A method, in a data processing system, for generating a minimized call tree data structure from trace data obtained from a plurality of executions of a computer program, comprising: obtaining a plurality of call tree data structures (col. 2, lines 32-33 "a call stack...one or more nodes in the tree") corresponding to the trace data (col. 2, lines 28-29 "trace information...obtained...") for the plurality of executions of the computer program (col. 2, lines, 38-39 "...number of Java bytecodes executed in each method... called").

Alexander(1) does not disclose generating a minimized call tree data structure from the plurality of call tree data structures wherein the minimized call tree data structure includes a minimum set of nodes that are consistent between the plurality of call tree data structures; and outputting the minimized call tree data structure.

However Kazi discloses in an analogous computer system generating a minimized call tree data structure from the plurality of call tree data structures (Kazi page 100 "Tree generation... merged trace files to create an output file containing the dynamic execution tree") wherein the minimized call tree data structure includes a minimum set of nodes that are consistent between the plurality of call tree data structures (Kazi page 100 "Tree generation... Run-time statistics generation...Each detailed... trace file is analyzed to gather the total number of calls made to each method, the maximum, minimum, and average execution times, and the standard deviation of the execution time for each method"). Thereby minimizing the display for execution and outputting the minimized call tree data structure (Kazi page 100 "Tree generation... merged trace files to create an output file containing the dynamic execution tree").

Therefore, it would have been obvious to a person of ordinary skill in the art at the time the invention was made to incorporate the method of generating a minimized call tree data structure from the plurality of call tree data structures wherein the minimized call tree data structure includes a minimum set of nodes that are consistent between the plurality of call tree data structures; and outputting the minimized call tree data structure as taught by Kazi into the method for providing the trace information as taught by Alexander(1). The modification would be obvious because of one of ordinary skill in the art would be motivated to generate a minimize call tree data structure from the trace data to provide a performance analysis tool to allow developer to determine the execution times have high of low variance as suggested by Kazi (page 115, “Conclusion”).

**Per claim 4:**

The rejection of claim 1 is incorporated and further, Alexander discloses:

4. The method of claim 1, wherein generating the minimized call tree data structure includes: copying a first call tree data structure; and walking a second call tree data structure over the first call tree data structure to generate the minimized call tree data structure (col. 6, lines 27-29 "the tree is traversed (i.e., walking) to the parent (using the parent pointer), and the current tree node is set equal to the parent node (step 178)… the tree can be dynamically pruned in order to reduce the amount of memory dedicated to its maintenance (step 179).").

**Per claim 5:**

The rejection of claim 4 is incorporated and further, Alexander discloses:

5. The method of claim 4, wherein walking the second call tree data structure over the first call tree data structure includes: for each node that exists in both the first call tree data structure and the second call tree data structure, generating a node in the minimized call tree data structure (col. 6, lines 33-35 "a check is made to determine if the module is already a child node of the current tree node (step 180). If not, a new node is created for the module and it is attached to the tree below the current tree node (step 182).") and associating values with the node (col. 7, lines 51-55 "Calls 190 lists the number of times each routine has been called. Base 192 is the total time spent in the routine. Cum 194 is the cumulative time spent in the routine and all routines called by the routine. Cum2196 is the cumulative time plus time spent in recursive routines" and FIG. 9A where the base cum and sum values are disclosed).

**Per claim 6:**

The rejection of claim 5 is incorporated and further, Alexander discloses:

6. The method of claim 5, wherein the values associated with the node are values that correspond to the minimum of the values associated with corresponding nodes in the first call tree data structure and the second call tree data structure (col. 7, lines 51-55 "Calls 190 lists… number of times each routine has been called. Base 192 is the total time spent in the routine. Cum 194 is the cumulative time spent in the routine and all routines called by the routine. Cum2196 is the cumulative time plus time spent in recursive routines" and FIG. 9A where the base cum and sum values are disclosed).

**Per claim 7:**

The rejection of claim 4 is incorporated and further, Alexander discloses:

7. The method of claim 4, wherein walking the second call tree data structure over the first call tree data structure includes: for each node that exists in only one of the first call tree data structure and the second call tree data structure, inhibiting creating a node in the minimum call tree data structures (col. 6, lines 33-35 "a check is made to determine if the module is already a child node of the current tree node (step 180). If not, a new node is created for the module and it is attached to the tree below the current tree node (step 182).").

**Per claim 8:**

The rejection of claim 6 is incorporated and further, Alexander discloses:

8. The method of claim 6, wherein the values associated with each node in the minimized call tree data structure include a minimum base value, a minimum number of calls, a minimum

cumulative value, and a minimum absolute cumulative value (col. 4, lines 43-48 "statistics shown include the number of distinct times the call stack is produced, the sum of the time spent in the call stack, the total time spent in the call stack plus the time in those call stacks invoked from this call stack (referred to as cumulative time), and the number of instances of this routine above this instance (indicating depth of recursion)").

**(10) Response to Argument**

**Appellant argued that:**

In the present case, the Examiner has not established a *primafacie* case of obviousness in rejecting the claims because neither Alexander nor Kazi nor Alexander in view of Kazi teaches or suggests all the claim limitations. With respect to claim 1, for example, neither Alexander nor Kazi nor Alexander in view of Kazi teaches or suggests "obtaining a plurality of call tree data structures corresponding to the trace data for the plurality of executions of the computer program", or "generating a minimized call tree data structure from the plurality of call tree data structures, wherein the minimized call tree data structure includes a minimum set of nodes that are consistent between the plurality of call tree data structures."

**Examiner's response:**

With respect to comments made by applicants for establishing a prima facie case of obviousness based up on prior art that Alexander and Kazi are nonanalogous art (remarks page 6), it has been held that a prior art reference must either be in the field of applicant's endeavor or, if not, then be reasonably pertinent to the particular problem with which the applicant was concerned, in order to be relied upon as a basis for rejection of the claimed invention. See *In re Oetiker*, 977 F.2d 1443, 24 USPQ2d 1443 (Fed. Cir. 1992). In this case, both Alexander and Kazi are analogous art. Alexander discloses the reduction of trace files which are taken in real time to improve the performance of the system (see Abstract and Summary). The reduction takes

by inputting the trace files, the trace information are obtained either statically or dynamically and represented as a tree of events. Kazi in a similar manner discloses the generation of trace execution threads of Java Virtual Machine. The Implantation is done by producing dynamic execution tree of traces (See, pages 99).

With respect to applicants argument that neither Alexander nor Kazi teach or suggest "obtaining a plurality of call tree data structures corresponding to the trace data for the plurality of executions of the computer program", or "generating a minimized call tree data structure from the plurality of call tree data structures, wherein the minimized call tree data structure includes a minimum set of nodes that are consistent between the plurality of call tree data structures." Examiner respectfully disagrees. Alexander discloses dynamically tracing and reduction, in which case information is obtained in real-time, as each event occurs, and is automatically reduced and added to the trace representation. Alternately, the tracing and reduction may be static, in which case a trace text file or binary file is obtained from a trace buffer, and the reduction takes place using the trace file as input, see col. 2, lines, 15-27.

More particularly, Kazi teaches Java class files are executed and execution trace files are generated, the tree representation as shown in figure 2. These trace files are then processed to generate dynamic execution tree. The trace files are merged to generate the execution call graph tree (page 99, first paragraph) and dynamic execution tree is displayed. The displayed call tree graph contains the parent node (root node) and child nodes (sub node) structure. The visualizer traverses the dynamic execution tree to search for the matching sub node and merges the sub nodes together, i.e., minimizing the call tree with minimum set of nodes. This merging and

matching of sub node is performed until the entire tree is searched (page 111) thus, call tree is generated with minimum set of nodes as shown in figure 13.

Here Kazi teaches merging of trace files to create an output file that contains the dynamic execution tree. Merging here is used to minimize the two files into one by combining the matching entries of the trace calls as explained above. Further, Kazi teaches minimizing the trace record by using the filtering option (page 102). Therefore, in regard to the merge step cited, it is clear that Kazi's discloses averaging specific information of a call, graph by totaling the data and computing the average. Averaging the various runs (various values of a respective node) of a computer program inherently minimizes variations (each value relating to a run) in the profile data (trace data).

Further, Kazi teaches "To facilitate the performance analysis of the call graph, statistical information about each of the methods in the program is gathered in the merge step. Each detailed .prf trace file is analyzed to gather the total number of calls made to each method, the maximum, minimum, and average execution times, and the standard deviation of the execution time for each method." (See Kazi, page 100, "Run-time statistics generation"). Hence, Kazi discloses generating a minimized call tree data structure from the plurality of call tree data structures (page 100 "Tree generation... merged trace files to create an output file containing the dynamic execution tree") wherein the minimized call tree data structure includes a minimum set of nodes that are consistent between the plurality of call tree data structures (page 100 "Tree generation... Run-time statistics generation...Each detailed... trace file is analyzed to gather the total number of calls made to each method, the maximum, minimum, and average execution times, and the standard deviation of the execution time for each method"). Thereby minimizing

the display for execution and outputting the minimized call tree data structure (page 100 "Tree generation... merged trace files to create an output file containing the dynamic execution tree").

Alexander discloses obtaining a plurality of call tree data structures (col. 2, lines 32-33 "a call stack...one or more nodes in the tree") corresponding to the trace data (col. 2, lines 28-29 "trace information...obtained...") for the plurality of executions of the computer program (col. 2, lines, 38-39 "...number of Java bytecodes executed in each method... called"). Applicants indicated that the Alexander disclose a single execution of a long running program. Examiner respectfully disagrees. Alexander discloses profiling the performance characteristic of long running programs/applications (col. 2, lines 56-57; col. 6, lines 50-55; col. 6, lines 60-65); note here Alexander uses the word programs as plural form which represents one or more programs, thus the tracing is performed on a plurality of programs execution.

Thus, the Alexander in combination with Kazi discloses "obtaining a plurality of call tree data structures corresponding to the trace data for the plurality of executions of the computer program", or "generating a minimized call tree data structure from the plurality of call tree data structures, wherein the minimized call tree data structure includes a minimum set of nodes that are consistent between the plurality of call tree data structures." Consequently, the amalgamation of both references is proper.

With respect to applicant's argument for claim 4 that Alexander does not teach or suggest generating a minimized call tree data structure. However, claim 4 is dependent on claim 1, and the rejection of claim 1 is incorporated into claim 4, examiner described as explained above that

the generating a minimized call tree is taught by Kazi and in the rejection. Further, Alexander discloses dynamically pruning the tree by traversing/walking the tree. The tree is traversed to the parent (using the parent pointer), and the current tree node is set equal to the parent node (step 178). At this point, the tree can be dynamically pruned (i.e., minimized) in order to reduce the amount of memory dedicated to its maintenance (step 179). If the trace event is an enter event, a check is made to determine if the module is already a child node of the current tree node (step 180). If not, a new node is created for the module and it is attached to the tree below the current tree node (step 182). The tree is then traversed to the module's node, and the current tree node is set equal to the module node (step 184). The number of calls to the current tree node is then incremented (step 186) (col. 6, lines 25-42). Thus, Alexander discloses walking/traversing the tree to prune (to minimize) the tree (col. 6, lines 62-63; col. 7, lines 3-16; col. 7, lines 45-46).

**(11) Related Proceeding(s) Appendix**

No decision rendered by a court or the Board is identified by the examiner in the Related Appeals and Interferences section of this examiner's answer.

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

Satish Rampuria
/Satish Rampuria/
Examiner, Art Unit 2191

Conferees:
Wei Zhen (SPE)
/Wei Y Zhen/
Supervisory Patent Examiner, Art Unit 2191

/Lewis A. Bullock, Jr./
Supervisory Patent Examiner, Art Unit 2193